****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Feb 24, 2024]** | **[Victor Udeh]** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

[Victor Udeh]

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

[**AES (Advanced Encryption Standard)** is the encryption algorithm cipher recommended for securing Artemis Financial’s software application. AES is a symmetric key encryption cipher, which means it uses the same key for both encrypting and decrypting information, ensuring a secure and efficient method for data protection. It operates on 128-bit blocks of data and supports key sizes of 128, 192, or 256 bits, offering a strong level of security and making it resilient against brute force attacks.

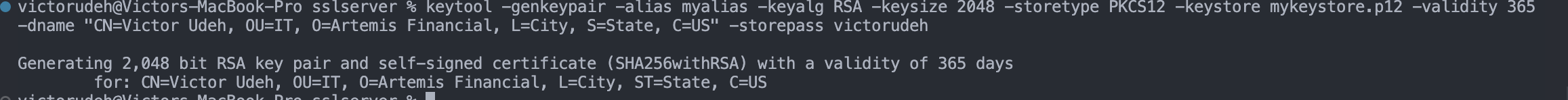
The use of random numbers in AES is crucial for generating secure keys. Secure random number generators ensure that the keys are unpredictable and safe from attackers. The distinction between symmetric (like AES) and non-symmetric (asymmetric) keys, such as RSA, lies in their use cases: AES is often used for encrypting data due to its speed and efficiency, while asymmetric keys are typically used for secure key exchange and digital signatures.

Historically, AES was established as a standard by the U.S. National Institute of Standards and Technology (NIST) in 2001, succeeding the older DES (Data Encryption Standard) due to DES's vulnerability to brute-force attacks. Today, AES is widely accepted and used across various industries, including government and financial institutions, highlighting its reliability and effectiveness in securing sensitive data.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

[Insert screenshot(s) here.]  
A screenshot of a computer

Description automatically generated

## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

[Insert screenshot(s) here.]  
A close-up of a number

Description automatically generated

## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

* Insert a screenshot below of the web browser that shows a secure webpage.

[Insert screenshot(s) here.]  
this is because it is self assignedA screenshot of a computer

Description automatically generated

## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

[Insert screenshots here.]A screen shot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

[Insert screenshot(s) here.]

A screen shot of a computer

Description automatically generated

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

[Include your findings here.]  
In refactoring Artemis Financial’s software application, AES encryption was deployed to enhance data security, transitioning communication protocols from HTTP to HTTPS to ensure secure data transmission. The application underwent secondary testing with OWASP Dependency-Check to identify and mitigate any introduced vulnerabilities, coupled with thorough functional testing to validate the implementation and maintain software integrity.

The process highlighted critical areas of security improvement, addressing vulnerabilities through encryption and secure communication protocols, thereby adding robust layers of security. Adhering to best practices in secure coding and regular vulnerability assessments has significantly elevated the application’s defense mechanisms, offering substantial value to Artemis Financial’s commitment to safeguarding sensitive financial data.